Abdelrahman Mohamed Ali

abdelrahmansalby23@gmail.com

Testbench code:

```
import ALSUPACKAGE::*;
module testbench ();
logic clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
opcode_e opcode;
logic signed [2:0] A, B;
logic [15:0] leds_dut, leds_ref;
logic [5:0] out_dut, out_ref;
logic innvalid;
ALSUtransaction tr = new();
ALSU DUT(
    A, B, cin, serial_in, red_op_A, red_op_B, opcode,
    bypass_A, bypass_B, clk, rst, direction, leds_dut, out_dut
);
ALUS_ref REF(
    A, B, cin, serial_in, red_op_A, red_op_B,
    opcode, bypass_A, bypass_B, clk, rst, direction, leds_ref, out_ref
);

initial begin
    clk = 0;
    forever
        #1 clk = ~clk;
end

assign innvalid = REF.invalid;
initial begin
    // First loop turn off constraint 8
    tr.rules_8.constraint_mode(0);
    repeat(100_000) begin
        randomization;
        @(posedge clk);
        sample;
    end

    // Forced rst
    bypass_A = 0; bypass_B = 0;
    red_op_A = 0; red_op_B = 0;
    rst = 1; @(posedge clk); rst = 0;

    // Seconed loop turn ON constraint 8 and turn OFF other constraints
    tr.rules1_7.constraint_mode(0);
    tr.rules_8.constraint_mode(1);
    repeat(10_000) begin
        randomization;
        foreach(tr.arr[i])
            opcode = tr.arr[i];
        $display("arr = %p",tr.arr);
        @(posedge clk);
        sample;
    end
    $stop;
end

task randomization;
    assert(tr.randomize());
    rst = tr.rst;
    cin = tr.cin;
    red_op_A = tr.red_op_A;
    red_op_B = tr.red_op_B;
    bypass_A = tr.bypass_A;
    bypass_B = tr.bypass_B;
    direction = tr.direction;
    serial_in = tr.serial_in;
    opcode = tr.opcode;
    A = tr.A;
    B = tr.B;
endtask

task sample;
    if (~tr.rst||~bypass_A||~bypass_B) tr.cvr_gp.sample();
endtask

assert_invaled:         assert property (@(posedge clk) REF.invalid |-> DUT.invalid);
assert_invaled_opcode:  assert property (@(posedge clk) REF.invalid_c1 |-> DUT.invalid_opcode);
assert_invaled_ref_op:  assert property (@(posedge clk) REF.invalid_c2 |-> DUT.invalid_red_op);
assert_out:             assert property (@(posedge clk) (out_dut == out_ref));

endmodule
```

Package code:

```
package ALSUPACKAGE;
    typedef enum reg [2:0] {OR, XOR, ADD, MULT, SHIFT, ROTATE, INVALID_6, INVALID_7} opcode_e;
    parameter MAXPOS = 3;
    parameter ZERO = 0;
    parameter MAXNEG = -4;

class ALSUtransaction;
    bit first_rst = 0;
    rand opcode_e opcode;
    rand bit rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
    rand bit signed [2:0] A, B;
```

/// ---------- Constraint ---------- ///

```
    constraint rules1_7 {
        //rst constraint
            rst dist {0:=95, 1:=5};

        // Invalid cases constraint
            opcode dist {INVALID_6:=5, INVALID_7:=5, [0:5]:/90};

        // A & B constraint when opcode is ADD or MULT
            (opcode == MULT || opcode == ADD) -> A dist {MAXPOS:=20, ZERO:=10, MAXNEG:=20, [MAXNEG+1:MAXPOS-1]:/50};
            (opcode == MULT || opcode == ADD) -> B dist {MAXPOS:=20, ZERO:=10, MAXNEG:=20, [MAXNEG+1:MAXPOS-1]:/50};

        // A & B constraint when opcode is OR or XOR and red_op_A is high
            ((opcode==XOR || opcode==OR) && red_op_A) -> A dist {3'b001:=30, 3'b010:=30, 3'b100:=30, [MAXNEG+1:MAXPOS-1]:/10};
            ((opcode==XOR || opcode==OR) && red_op_A) -> B == 0;

        // A & B constraint when opcode is OR or XOR and red_op_B is high
            ((opcode==XOR || opcode==OR) && red_op_B) -> A == 0;
            ((opcode==XOR || opcode==OR) && red_op_B) -> B dist {3'b001:=30, 3'b010:=30, 3'b100:=30, [MAXNEG+1:MAXPOS-1]:/10};

        //  Do not constraint the inputs A or B when the operation is shift or rotate
        // it's achieved by default after the 2,3 and 4 Constraint achieved

        // bypass constraint
            bypass_A dist {0:=90, 1:=10};
            bypass_B dist {0:=90, 1:=10};

        // red_op constraint
            red_op_A dist {0:=90, 1:=10};
            red_op_B dist {0:=90, 1:=10};

        // A & B constraint when opcode is OR or XOR and red_op is low
            ((opcode==XOR || opcode==OR) && ~red_op_A && ~red_op_B) -> A == ~B;
    }

    rand opcode_e arr [6];
    constraint rules_8 {
        foreach(arr[i])
            arr[i] inside {OR, XOR, ADD, MULT, SHIFT, ROTATE};
        unique {arr};
    }
```

/// ---------- Cover group ---------- ///

```
covergroup cvr_gp@(posedge clk);
    // input A bins
        A_cp: coverpoint A {
            bins A_data_0 = {ZERO};
            bins A_data_max = {MAXPOS};
            bins A_data_min = {MAXNEG};
            bins A_data_default = default;
            bins A_data_[] = {001, 010, 100};
        }

    // input B bins
        B_cp: coverpoint B {
            bins B_data_0 = {ZERO};
            bins B_data_max = {MAXPOS};
            bins B_data_min = {MAXNEG};
            bins B_data_default = default;
            bins B_data_[] = {001, 010, 100};
        }

    // cover point for reduction operation red_op
        op_A_cp: coverpoint red_op_A {
            bins one = {1};
            bins zero = {0};
            option.weight = 0;
        }
        op_B_cp: coverpoint red_op_B {
            bins one = {1};
            bins zero = {0};
            option.weight = 0;
        }

    // Crossing to satsfied the data_walkingones of A and B
        A_walk: cross A_cp, op_A_cp {
            option.cross_auto_bin_max = 0;
            bins A_data_walkingones = binsof(A_cp.A_data_) && binsof(op_A_cp.one);
        }
        B_walk: cross B_cp, op_A_cp, op_B_cp {
            option.cross_auto_bin_max = 0;
            bins B_data_walkingones = (binsof(B_cp.B_data_)
                                    && binsof(op_A_cp.zero)
                                    && binsof(op_B_cp.one));
        }

    // cover point for opcode (ALU)
        ALU_cp: coverpoint opcode {
            bins Bins_shift[] = {SHIFT, ROTATE};
            bins Bins_arith[] = {ADD, MULT};
            bins Bins_bitwise[] = {OR, XOR};//
            bins Bins_invalid = {INVALID_6, INVALID_7};
            bins Bins_trans = (0 => 1 => 2 => 3 => 4 => 5);
        }

    // cover point for c_in
        cin_cp: coverpoint cin {
            option.weight = 0;
        }

    // cover point for serial_in
        serial_cp: coverpoint serial_in{
            option.weight = 0;
        }

    // cover point for direction
        direction_cp: coverpoint direction{
            option.weight = 0;
        }

    // Cross coverage between ALU_cp and A and B
        ALU_A: cross ALU_cp, A_cp {
            option.cross_auto_bin_max = 0;
            bins arith_permutations = binsof(ALU_cp.Bins_arith) && binsof(A_cp) intersect{ZERO, MAXPOS, MAXNEG};
        }
        ALU_B: cross ALU_cp, B_cp {
            option.cross_auto_bin_max = 0;
            bins arith_permutations = binsof(ALU_cp.Bins_arith) && binsof(B_cp) intersect{ZERO, MAXPOS, MAXNEG};
        }

    // Cross coverage between ALU_cp and cin_cp
        ALU_cin: cross ALU_cp, cin_cp {
            option.cross_auto_bin_max = 0;
            bins add_cin = binsof(ALU_cp) intersect{ADD};
        }

    // Cross coverage between ALU_cp and serial_cp
        ALU_serial: cross ALU_cp, serial_cp {
            option.cross_auto_bin_max = 0;
            bins shift_serial = binsof(ALU_cp) intersect{SHIFT};
        }
```

```
        // Cross coverage between ALU_cp and direction_cp
            ALU_direction: cross ALU_cp, direction_cp {
                option.cross_auto_bin_max = 0;
                bins sh_ro_direction = binsof(ALU_cp.Bins_shift);
            }

        // Cross coverage ALU = {OR,XOR}, red_op_A = 1, A = data_walk, B = 0
            A_data_walk_OR_XOR: cross ALU_cp, A_walk, op_A_cp, B_cp {
                option.cross_auto_bin_max = 0;
                bins A_walk_OR_XOR = (binsof(ALU_cp.Bins_bitwise)
                                    && binsof(A_walk.A_data_walkingones)
                                    && binsof(op_A_cp.one)
                                    && binsof(B_cp.B_data_0));
            }

        // Cross coverage ALU = {OR,XOR}, red_op_A = 1, A = data_walk, B = 0
            B_data_walk_OR_XOR: cross ALU_cp, B_walk, op_B_cp, A_cp {
                option.cross_auto_bin_max = 0;
                bins B_walk_OR_XOR = (binsof(ALU_cp.Bins_bitwise)
                                    && binsof(B_walk.B_data_walkingones)
                                    && binsof(op_B_cp.one)
                                    && binsof(A_cp.A_data_0));
            }

        // Cross coverage Invalid case 2 red_op
            Invalid_red_op: cross ALU_cp, op_A_cp, op_B_cp {
                option.cross_auto_bin_max = 0;
                bins Invalid_reduction = (binsof(ALU_cp.Bins_bitwise)
                                        && (binsof(op_A_cp.one) || binsof(op_B_cp.one)));
            }

        // Invalid case with reduction operation
            reduction_invalid: cross ALU_cp, red_op_A, red_op_B {
                option.cross_auto_bin_max = 0;
                bins invalid_red_op = (~binsof(ALU_cp.Bins_bitwise) && (binsof(red_op_A.one)||binsof(red_op_B.one)));
            }

        // cover point rst
            rst_cp: coverpoint rst {
                option.weight = 0;
            }
    endgroup
```

```
    function new();
        cvr_gp = new();
    endfunction //new()
    function void post_randomize();
        if(!rst_chk) rst = 1;
        rst_chk = 1;
    endfunction
endclass //ALSUtransaction
endpackage
```

Design code:

/// ---------- DUT code ---------- ///

```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: ALSU Design
//
////////////////////////////////////////////////////////////////////////////////
module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;
output reg [15:0] leds;
output reg [5:0] out;

reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2:0] opcode_reg, A_reg, B_reg;

wire invalid_red_op, invalid_opcode, invalid;

assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
// assign invalid_opcode = opcode_reg[2] & opcode_reg[3]; // Wrong
assign invalid_opcode = opcode_reg[1] & opcode_reg[2]; // Fix
assign invalid = invalid_red_op | invalid_opcode;


always @(posedge clk or posedge rst) begin
  if(rst) begin
    leds <= 0;
    out <= 0;
    cin_reg <= 0;
    red_op_B_reg <= 0;
    red_op_A_reg <= 0;
    bypass_B_reg <= 0;
    bypass_A_reg <= 0;
    direction_reg <= 0;
    serial_in_reg <= 0;
    opcode_reg <= 0;
    A_reg <= 0;
    B_reg <= 0;

  end
  else begin
    if (invalid)
      leds <= ~leds;
    else
      leds <= 0;
    cin_reg <= cin;
    red_op_B_reg <= red_op_B;
    red_op_A_reg <= red_op_A;
    bypass_B_reg <= bypass_B;
    bypass_A_reg <= bypass_A;
    direction_reg <= direction;
    serial_in_reg <= serial_in;
    opcode_reg <= opcode;
    A_reg <= A;
    B_reg <= B;
    if (invalid)
        out <= 0;
    else if (bypass_A_reg && bypass_B_reg)
      out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
    else if (bypass_A_reg)
      out <= A_reg;
    else if (bypass_B_reg)
      out <= B_reg;
    else begin
        //case (opcode) // wrong
        case (opcode_reg) // FIX
          //3'h0: begin // Wrong
          //  if (red_op_A_reg && red_op_B_reg)
          //    out = (INPUT_PRIORITY == "A")? &A_reg: &B_reg;
          //  else if (red_op_A_reg)
          //    out <= &A_reg;
          //  else if (red_op_B_reg)
          //    out <= &B_reg;
          //  else
          //    out <= A_reg & B_reg;
          //end
          3'h0: begin // Fix
            if (red_op_A_reg && red_op_B_reg)
              out = (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
            else if (red_op_A_reg)
              out <= |A_reg;
            else if (red_op_B_reg)
              out <= |B_reg;
            else
              out <= A_reg | B_reg;
          end
```

```
        //3'h1: begin // Wrong
        //  if (red_op_A_reg && red_op_B_reg)
        //    out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
        //  else if (red_op_A_reg)
        //    out <= |A_reg;
        //  else if (red_op_B_reg)
        //    out <= |B_reg;
        //  else
        //    out <= A_reg | B_reg;
        //end
        3'h1: begin // Fix
          if (red_op_A_reg && red_op_B_reg)
            out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
          else if (red_op_A_reg)
            out <= ^A_reg;
          else if (red_op_B_reg)
            out <= ^B_reg;
          else
            out <= A_reg ^ B_reg;
        end
        3'h2: begin
          if (FULL_ADDER == "ON")
            out <= A_reg + B_reg + cin_reg;
          else
            out <= A_reg + B_reg;
        end
        3'h3: out <= A_reg * B_reg;
        3'h4: begin
          if (direction_reg)
            out <= {out[4:0], serial_in_reg};
          else
            out <= {serial_in_reg, out[5:1]};
        end
        3'h5: begin
          if (direction_reg)
            out <= {out[4:0], out[5]};
          else
            out <= {out[0], out[5:1]};
        end
      endcase
    end
  end
end

endmodule
```

/// ---------- Reference code ---------- ///

```
module ALUS_ref (A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
parameter INPUT_PRIORITY = "A";
parameter FULL_ADDER = "ON";
input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
input [2:0] opcode;
input signed [2:0] A, B;
output reg [15:0] leds;
output reg [5:0] out;

reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
reg [2:0] opcode_reg, A_reg, B_reg;

wire invalid, invalid_c1, invalid_c2;
assign invalid_c1 = (opcode_reg == 3'h6 || opcode_reg == 3'h7);
assign invalid_c2 = (red_op_A_reg || red_op_B_reg) && (opcode_reg != 3'h0 && opcode_reg != 3'h1);
assign invalid = invalid_c1 || invalid_c2;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        out <= 0;
        leds <= 0;
        cin_reg <= 0;
        red_op_A_reg <= 0;
        red_op_B_reg <= 0;
        bypass_A_reg <= 0;
        bypass_B_reg <= 0;
        direction_reg <= 0;
        serial_in_reg <= 0;
        opcode_reg <= 0;
        A_reg <= 0;
        B_reg <= 0;
    end
    else begin
        if (invalid) leds <= ~leds;
        else leds <= 0;

        cin_reg <= cin;
        red_op_A_reg <= red_op_A;
        red_op_B_reg <= red_op_B;
        bypass_A_reg <= bypass_A;
        bypass_B_reg <= bypass_B;
        direction_reg <= direction;
        serial_in_reg <= serial_in;
        opcode_reg <= opcode;
        A_reg <= A;
        B_reg <= B;

        if (invalid)
            out <= 0;
        else if (bypass_A_reg && bypass_B_reg)
            out <= (INPUT_PRIORITY=="A")? A_reg:B_reg;
        else if (bypass_A_reg)
            out <= A_reg;
        else if (bypass_B_reg)
            out <= B_reg;
        else if (opcode_reg == 3'h0) begin
            if (red_op_A_reg && red_op_B_reg)
                out <= (INPUT_PRIORITY=="A")? (|A_reg):(|B_reg);
            else if (red_op_A_reg)
                out <= |A_reg;
            else if (red_op_B_reg)
                out <= |B_reg;
            else
                out <= A_reg|B_reg;
        end
        else if (opcode_reg == 3'h1) begin
            if (red_op_A_reg && red_op_B_reg)
                out <= (INPUT_PRIORITY=="A")? (^A_reg):(^B_reg);
            else
                out <= (red_op_A_reg)? (^A_reg): (red_op_B_reg)? (^B_reg):(A_reg^B_reg);
        end
        else if (opcode_reg == 3'h2)
            out <= (FULL_ADDER=="ON")? (A_reg + B_reg + cin_reg):(A_reg + B_reg);
        else if (opcode_reg == 3'h3)
            out <= A_reg * B_reg;
        else if (opcode_reg == 3'h4) begin // SHIFT
            if (direction_reg)
                out <= {out[4:0], serial_in_reg};
            else
                out <= {serial_in_reg, out[5:1]};
        end
        else if (opcode_reg == 3'h5) begin // ROTATE
            if (direction_reg)
                out <= {out[4:0],out[5]};
            else
                out <= {out[0],out[5:1]};
        end
    end
end
endmodule
```

Snippet to your verification requirement document

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
ALSU_1	Incase of invalid cases do not occur, when opcode is add, then out should perform the addition on ports A and B taking cin if parameter FULL_ADDER is high	Randomization under constraints on the A and B to have the maximum, minimum and zero values most of the time	Included as coverpoint for A and B. Included with cross coverage when ALU opcode is addition or multiplication	Output Checked against golden model
ALSU_2	Incase of invalid cases do not occur, when opcode is mult, then out should perform the multiplication on ports A and B	Randomization under constraints on the A and B to have the maximum, minimum and zero values most of the time	Included in coverpoint2 for A and B. Included with cross coverage when ALU opcode is addition or multiplication	Output Checked against golden model
ALSU_3	When invalid cases exist, out output should be low and leds should blink	Randomization under constraints where invalid cases do not occur as frequent as valid cases	Included in a coverpoint for opcode. Included with cross coverage to make sure invalid cases occur	Output Checked against golden model
ALSU_4	If invalid cases do not occur and the bypass inputs are high, then the output out should by bypass port A or B based on the prioirty if the both bypass ports are high	Randomization for bypass	Included in a coverpoint for bypass	Output Checked against golden model
ALSU_5	If invalid cases do not occur and the reduction operation are high and opcode nither OR nor XOR, then the output out should by zero and leds should blink as an INVALED case	Randomization for reduction operation	Included in cover group a cross covarage for reduction Invalid (reduction_invalid) Included with cross coverage when ALU opcode is addition or multiplication	Output Checked against golden model and from invvalid signal
ALSU_6	Incase of invalid cases do not occur, when opcode is OR and reduction operation is low, then out should perform make A OR B	Randomization under constraints on the A to be invert B (A=~B)	Included in coverpoint for A and B. Included with cross coverage when ALU opcode is arithmatic	Output Checked against golden model
ALSU_7	Incase of invalid cases do not occur, when opcode is XOR and reduction operation is low, then out should perform A XOR B	Randomization under constraints on the A to be invert B (A=~B)	Included in coverpoint for A and B. Included with cross coverage when ALU opcode is arithmatic	Output Checked against golden model
ALSU_8	Incase of invalid cases do not occur, when opcode either OR or XOR and reduction operation is high for port A, then out should perform reduction OR or XOR on ports A onley	Randomization under constraint the input B most of the time to have one bit high in its 3 bits while constraining the A to be low	Included in coverpoint for A. when A is {001, 010, 100} and only the red_op_A is high	Output Checked against golden model
ALSU_9	Incase of invalid cases do not occur, when opcode either OR or XOR and reduction operation is high for port B, then out should perform reduction OR or XOR on ports B onley	Randomization under constraint the input A most of the time to have one bit high in its 3 bits while constraining the B to be low	Included in coverpoint for A. when B is {001, 010, 100} and onley red_op_A is low and red_op_B is high	Output Checked against golden model
ALSU_10	Incase of invalid cases do not occur, when opcode is SHIFT and direction is low, then out should perform shift left logical for last out with serial in	Randomization for A and B with no constraint	Included in coverpoint2 for A and B. Included with cross coverage when ALU opcode is SHIFT or ROTATE	Output Checked against golden model
ALSU_11	Incase of invalid cases do not occur, when opcode is SHIFT and direction is high, then out should perform shift right logical for last out with serial in	Randomization for A and B with no constraint	Included in coverpoint2 for A and B. Included with cross coverage when ALU opcode is SHIFT or ROTATE	Output Checked against golden model
ALSU_12	Incase of invalid cases do not occur, when opcode is ROTATE and direction is high, then out should perform the MSB to be LSB	Randomization for A and B with no constraint	Included in coverpoint2 for A and B. Included with cross coverage when ALU opcode is SHIFT or ROTATE	Output Checked against golden model
ALSU_13	Incase of invalid cases do not occur, when opcode is ROTATE and direction is low, then out should perform the LSB to be MSB	Randomization for A and B with no constraint	Included in coverpoint2 for A and B. Included with cross coverage when ALU opcode is SHIFT or ROTATE	Output Checked against golden model
ALSU_14	rst is high	Randomization under constraints on rst to be low most of time and make the first randomized value is high in post_randomize	include in coverpoint with weight zero	Output Checked from wave
ALSU_15	making opcode always valid	randomize an array and make it unique with no INVALID value	included in coverpoint ALU_cp	Output Checked against golden model

Do file

```
vlib work

vlog -coveropt 3 +cover +acc package.sv ALSU.v ALUS_ref.v testbench.sv

vsim -voptargs=+acc work.testbench

vsim -coverage -vopt work.testbench -c -do "add wave *; coverage save -onexit -du ALSU -directive -codeAll cover.ucdb; run -all"

coverage report -detail -cvg -directive -comments -output {F:/digital verification/Session 4/New folder/FUNCTION_COVER_ALSU.txt} {}

quit -sim

vcover report cover.ucdb -details -all -annotate -output cover_alsu.txt

vcover report -html cover.ucdb
```

Code Coverage & Functional Coverage report snippets

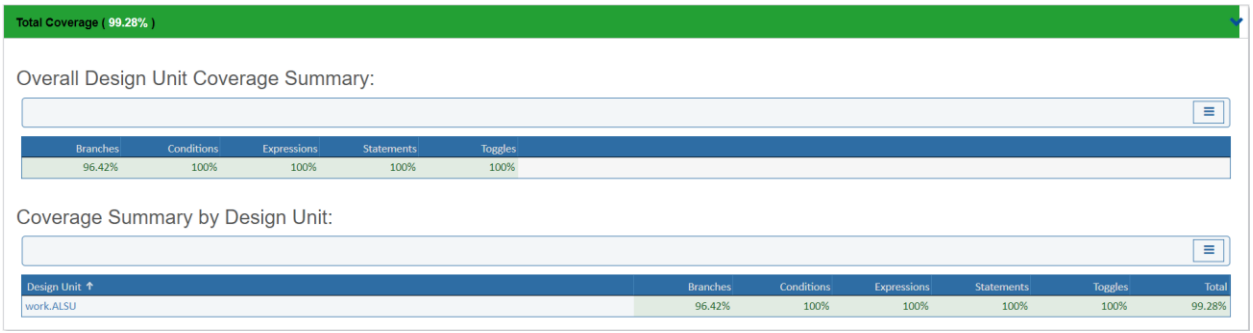
```
Coverage Report by instance with details

================================================================================
=== Instance: /\testbench#DUT
=== Design Unit: work.ALSU
================================================================================
Branch Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Branches                       28        27         1    96.42%
```

```
Condition Coverage:
    Enabled Coverage             Bins   Covered    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Conditions                      6         6         0   100.00%
```

```
Expression Coverage:
    Enabled Coverage             Bins   Covered    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Expressions                     8         8         0   100.00%
```

```
Statement Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Statements                     46        46         0   100.00%
```

```
Toggle Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Toggles                       118       118         0   100.00%
```



Total Coverage (99.28%)

Overall Design Unit Coverage Summary:

Branches	Conditions	Expressions	Statements	Toggles
96.42%	100%	100%	100%	100%

Coverage Summary by Design Unit:

Design Unit	Branches	Conditions	Expressions	Statements	Toggles	Total
work.ALSU	96.42%	100%	100%	100%	100%	99.28%

! NOTE: Branch coverage can’t be reached because of case default;

```
------------------------------------CASE Branch-----------------------------------
    67                                  65181     Count coming in to CASE
    78              1                   18347               3'h0: begin // Fix
    98              1                   11038               3'h1: begin // Fix
    108             1                    8844               3'h2: begin
    114             1                    9012               3'h3: out <= A_reg * B_reg;
    115             1                    9006               3'h4: begin
    121             1                    8934               3'h5: begin
                                        ***0***     All False Count
Branch totals: 6 hits of 7 branches = 85.71%
```

/// **---------- Coverage report ----------** *///*

```
Coverage Report by instance with details

================================================================================
=== Instance: /ALSUPACKAGE
=== Design Unit: work.ALSUPACKAGE
================================================================================

Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses        17        na        na        na
            Covergroup Bins        35        35         0   100.00%
```

```
Covergroup instance \/ALSUPACKAGE::ALSUtransaction::cvr_gp
                                              100.00%        100          -    Covered
   covered/total bins:                             35         35          -
   missing/total bins:                              0         35          -
   % Hit:                                      100.00%        100          -
   Coverpoint A_cp                             100.00%        100          -    Covered
       covered/total bins:                           4          4          -
       missing/total bins:                           0          4          -
       % Hit:                                  100.00%        100          -
       bin A_data_0                                17330          1          -    Covered
       bin A_data_max                              15510          1          -    Covered
       bin A_data_min                              16441          1          -    Covered
       bin A_data_[1]                              12705          1          -    Covered
       default bin A_data_default                  48015                     -    Occurred
   Coverpoint B_cp                             100.00%        100          -    Covered
       covered/total bins:                           4          4          -
       missing/total bins:                           0          4          -
       % Hit:                                  100.00%        100          -
       bin B_data_0                                17636          1          -    Covered
       bin B_data_max                              15402          1          -    Covered
       bin B_data_min                              16301          1          -    Covered
       bin B_data_[1]                              12657          1          -    Covered
       default bin B_data_default                  48005                     -    Occurred
```

Coverpoint op_A_cp [1]	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin one	14971	1	-	Covered
bin zero	95030	1	-	Covered
Coverpoint op_B_cp [1]	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin one	14738	1	-	Covered
bin zero	95263	1	-	Covered
Coverpoint ALU_cp	100.00%	100	-	Covered
covered/total bins:	8	8	-	
missing/total bins:	0	8	-	
% Hit:	100.00%	100	-	
bin Bins_shift[SHIFT]	16246	1	-	Covered
bin Bins_shift[ROTATE]	16373	1	-	Covered
bin Bins_arith[ADD]	16091	1	-	Covered
bin Bins_arith[MULT]	16263	1	-	Covered
bin Bins_bitwise[OR]	16355	1	-	Covered
bin Bins_bitwise[XOR]	16170	1	-	Covered
bin Bins_invalid	12503	1	-	Covered
bin Bins_trans	1	1	-	Covered

Coverpoint cin_cp [1]	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin auto[0]	55276	1	-	Covered
bin auto[1]	54725	1	-	Covered
Coverpoint serial_cp [1]	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin auto[0]	55213	1	-	Covered
bin auto[1]	54788	1	-	Covered
Coverpoint direction_cp [1]	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin auto[0]	55014	1	-	Covered
bin auto[1]	54987	1	-	Covered
Cross A_walk	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin A_data_walkingones	2309	1	-	Covered

Cross B_walk	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin B_data_walkingones	1865	1	-	Covered
Cross ALU_A	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin arith_permutations	18328	1	-	Covered
Cross ALU_cin	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin add_cin	16091	1	-	Covered
Cross ALU_serial	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin shift_serial	16246	1	-	Covered

Cross ALU_direction	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin sh_ro_direction	32619	1	-	Covered
Cross A_data_walk_OR_XOR	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin A_walk_OR_XOR	2922	1	-	Covered
Cross B_data_walk_OR_XOR	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin B_walk_OR_XOR	2888	1	-	Covered
Cross Invalid_red_op	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin Invalid_reduction	7433	1	-	Covered

```
 [1] - Does not contribute coverage as weight is 0

TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

Total Coverage By Instance (filtered view): 100.00%
```

Bugs reported if any:

1. Line 65
   case that check opcode must check opcode_reg Not opcode
   Error:
   ```
   case (opcode) ...
   endcase
   ```
   Fix:
   ```
   case (opcode_reg) ...
   endcase
   ```

2. Line 23
   the opcode_reg[3] is out of range
   Error:
   ```
   assign invalid_opcode = opcode_reg[2] & opcode_reg[3];
   ```
   Fix:
   ```
   assign invalid_opcode = opcode_reg[2] & opcode_reg[1];
   ```

3. Line 68
   when opcode_reg == 0'h0 it means the operation is OR not AND
   Error:
   ```
   3'h0: begin
       if (red_op_A_reg && red_op_B_reg)
           out = (INPUT_PRIORITY == "A")? &A_reg: &B_reg;
       else if (red_op_A_reg)
           out <= &A_reg;
       else if (red_op_B_reg)
           out <= &B_reg;
       else
           out <= A_reg & B_reg;
   end
   ```

   Fix:
   ```
   3'h0: begin
       if (red_op_A_reg && red_op_B_reg)
           out = (INPUT_PRIORITY == "A")? &A_reg: &B_reg;
       else if (red_op_A_reg)
           out <= &A_reg;
       else if (red_op_B_reg)
           out <= &B_reg;
       else
           out <= A_reg & B_reg;
   end
   ```

4. Line 88
   when opcode_reg == 0'h1 it means the operation is XOR not OR
   Error:
   ```
   3'h1: begin
       if (red_op_A_reg && red_op_B_reg)
           out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg;
       else if (red_op_A_reg)
           out <= |A_reg;
       else if (red_op_B_reg)
           out <= |B_reg;
       else
           out <= A_reg | B_reg;
   end
   ```

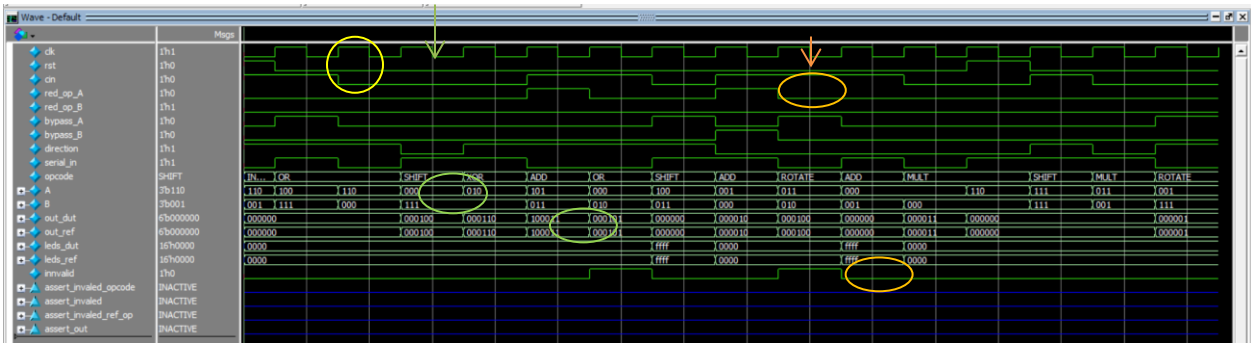
   Fix:
   ```
   3'h1: begin
       if (red_op_A_reg && red_op_B_reg)
           out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
       else if (red_op_A_reg)
           out <= ^A_reg;
       else if (red_op_B_reg)
           out <= ^B_reg;
       else
           out <= A_reg ^ B_reg;
   end
   ```

QuestaSim snippet:

rst = 1; expected out = 0, leds = 0; YELLOW

opcode = OR; A = 110; B = 000; expected output = 00110; LIGHT GREEN

opcode = ADD; red_op_A is high; expected output = 0; INVALID ORANGE

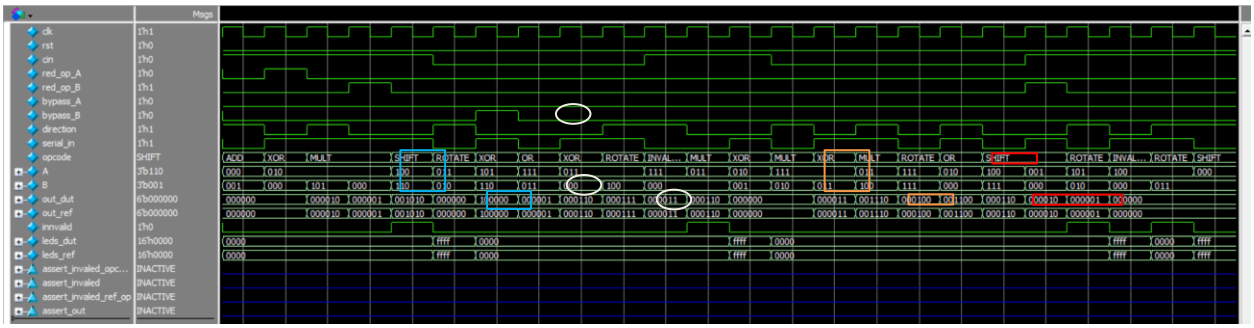


opcode = MULT; A =010; B = 101; expected out = 001010; LIGHT BLUE

opcode = XOR; B = 110; bypass_B is high; expected out = B; WHITE

opcode = XOR; A = 010; B = 001; expected out = 011; ORANGE

opcode = ROTATE; direction is low; last out = 001100; expected out = 000110; RED



opcode = ADD; A = 011; B = 101; cin is high; expected out =001001; YELLOW

opcode = SHIFT; serial_in = 1; direction is low LS L; last out = 100011; expected out = 110001; LIGHT BLUE

opcode = INVALID; RED